Respiratory Disease Diagnostic Assistant: A RAG-Based Approach

Abstract

Chronic respiratory diseases affect over one billion people globally, yet accurate diagnosis remains challenging due to overlapping symptoms and limited clinical resources. This project develops a diagnostic support tool using retrieval-augmented generation (RAG) to assist clinicians in identifying respiratory diseases based on patient symptoms. Using a dataset of 37,000+ patient records from the Albuquerque Public Health Department, the system implements a two-stage workflow: first, patient data is embedded and stored in a vector database; second, user queries retrieve similar cases and generate diagnostic suggestions through a conversational AI interface. This approach provides transparent, case-based reasoning that can be updated with new data without retraining, offering a practical entry point into intelligent clinical decision support.

1. Introduction

1.1 Problem Background

Respiratory diseases including asthma, COPD, tuberculosis, and pulmonary hypertension present similar symptoms that complicate diagnosis. Shortness of breath, coughing, and wheezing appear across multiple conditions, making it difficult for clinicians to quickly narrow down the correct diagnosis, especially in resource-constrained settings. Traditional diagnostic methods rely on clinical experience and testing, which may not always be immediately available.

1.2 Project Goals

This project aims to create a practical diagnostic assistant that:

- Accepts symptom descriptions and patient demographics as input
- Retrieves similar historical patient cases from a database
- Provides a single most likely disease diagnosis with explanation
- References actual patient cases to justify its recommendation
- Updates easily as new patient data becomes available

1.3 Why RAG?

Rather than training a traditional machine learning model that requires extensive data processing and retraining, this project uses retrieval-augmented generation. RAG combines database search with AI reasoning: it finds relevant historical cases and uses an AI model to interpret them. This approach offers transparency (showing which cases informed the diagnosis) and flexibility (adding new cases to the database without retraining).

2. Background and Related Work

2.1 Clinical Decision Support

Clinical decision support systems help healthcare providers make diagnostic and treatment decisions. Early systems used rule-based logic (e.g., "if fever AND cough, then consider pneumonia"), which worked for simple cases but struggled with complexity. Machine learning approaches improved pattern recognition but often function as "black boxes" that don't explain their reasoning—a significant limitation in medical settings where clinicians need to understand and verify AI suggestions.

2.2 Retrieval-Augmented Generation

RAG represents a newer approach to AI systems. Instead of encoding all knowledge within a neural network, RAG systems:

1. Store information in a searchable database
2. Retrieve relevant information when queried
3. Use an AI model to reason about the retrieved information

For medical applications, this means the system can show clinicians which historical cases it consulted when making a suggestion, making the reasoning process transparent and verifiable.

2.3 Semantic Embeddings

Modern AI can convert text into numerical representations called embeddings that capture meaning. Similar concepts have similar embeddings—for example, "shortness of breath" and "dyspnea" would be positioned close together in the embedding space. This allows the system to find semantically similar patient cases even when different words are used to describe the same symptoms.

3. Data Description

3.1 Dataset Overview

The project uses a dataset from the Albuquerque Public Health Department containing over 37,000 anonymized patient records related to respiratory illnesses. The dataset is updated quarterly, providing ongoing access to current disease patterns.

3.2 Data Fields

Each patient record contains:

- **Symptoms**: Text description of patient symptoms
- **Age**: Patient age (numeric)
- **Sex**: Patient sex (categorical)
- **Nature**: Whether the illness is acute or chronic (categorical)
- **Disease**: Confirmed diagnosis (target variable)
- **Treatment**: Treatment provided (text)

3.3 Data Challenges

The dataset presents several practical challenges:

- **Overlapping symptoms**: The same symptom appears in multiple diseases
- **Multiple symptoms per patient**: Most patients present with several symptoms simultaneously
- **Varying descriptions**: Clinicians may describe similar symptoms differently
- **Class imbalance**: Common diseases like asthma appear more frequently than rare conditions
- **Missing information**: Some records may have incomplete data

3.4 Exploratory Analysis

Before building the system, analysis will examine:

- Which symptoms most commonly appear with each disease
- How symptom combinations differ across diseases
- Age and sex patterns for different respiratory conditions
- How frequently different diseases appear in the dataset

4. System Architecture

4.1 Overall Design

The system uses a two-pipeline workflow that separates data preparation (done once or periodically) from query processing (done in real-time when users make requests).

4.2 Pipeline 1: Data Preparation

This offline pipeline prepares the patient database for searching:

Step 1 - Data Loading: Patient records are loaded from storage in their original format.

Step 2 - Data Preprocessing: Records are cleaned and standardized. Symptom text is normalized (lowercased, consistent formatting), missing values are handled, and categorical fields are encoded.

Step 3 - Embedding Generation: Each patient record is converted into a numerical vector (embedding) that captures its semantic meaning. This embedding represents the combination of symptoms, demographics, and disease characteristics.

Step 4 - Vector Database Storage: Embeddings are stored in a vector database optimized for similarity search. The database indexes these vectors so similar cases can be retrieved quickly. Metadata (age, sex, disease, nature) is stored alongside embeddings to enable filtering.

4.3 Pipeline 2: Query Processing

This online pipeline handles diagnostic queries in real-time:

Step 1 - User Input: A clinician enters patient symptoms and demographics through a conversational interface (chat).

Step 2 - Query Embedding: The user's symptom description is converted into an embedding using the same model used for the database records.

Step 3 - Similarity Search: The system searches the vector database to find patient records with embeddings most similar to the query embedding. This retrieves cases with similar symptom presentations.

Step 4 - Metadata Filtering (optional): Results can be filtered based on patient demographics (e.g., only retrieve cases from patients of similar age or sex).

Step 5 - Context Assembly: Retrieved cases are formatted as context, including their symptoms and confirmed diagnoses.

Step 6 - AI Reasoning: A conversational AI model analyzes the retrieved cases and the user's query. It synthesizes information across similar cases and generates a diagnostic suggestion with explanation.

Step 7 - Response Generation: The system returns a single most likely disease along with reasoning that references the historical cases consulted.

4.4 Key Technical Components

Embedding Model: Converts text to vector representations. Must capture medical terminology and symptom relationships.

Vector Database: Stores embeddings and enables fast similarity search across thousands of records.

Conversational AI Model: Provides reasoning capabilities to interpret retrieved cases and generate explanations in natural language.

Workflow Platform: Coordinates the entire process, managing data flow between components.

5. Testing and Validation

5.1 Testing Approach

Since this is a pure RAG implementation without trained models, testing focuses on verifying that the end-to-end workflow functions correctly and produces useful results.

5.2 Functional Tests

Retrieval Quality: Do the retrieved cases actually match the input symptoms? Testing involves manually reviewing several queries to verify the system finds relevant historical cases.

Reasoning Quality: Does the AI model produce logical diagnostic suggestions based on the retrieved cases? Are explanations coherent and medically reasonable?

Consistency: Does the system give similar answers when the same symptoms are described using different words?

Edge Cases: How does the system handle unusual inputs like rare symptom combinations or missing demographic information?

5.3 Accuracy Testing

Using a held-out subset of the patient database:

1. Query the system with actual patient symptoms
2. Compare the system's suggested diagnosis to the confirmed diagnosis
3. Calculate the percentage of correct diagnoses

This provides a basic accuracy measure for how often the system identifies the correct disease.

5.4 Performance Testing

- **Response Time**: Measure how long the system takes from query to response. Target: under 10 seconds.
- **Reliability**: Test the workflow over extended periods to identify any component failures.

5.5 Minimum Viable Product

The working system must:

- Accept symptom descriptions and demographics via chat interface
- Retrieve relevant historical cases from the vector database
- Generate a single disease diagnosis with explanation
- Reference specific retrieved cases in its reasoning
- Operate stably without crashes or errors
- Respond within reasonable time

6. Deployment

6.1 Deployment Options

The project will evaluate two deployment approaches:

Option 1 - Integrated Workflow: Deploy within the workflow automation platform itself. This keeps all components in one environment with built-in monitoring and error handling.

Option 2 - Independent Hosting: Extract the system and host it separately, potentially as a web service. This provides more control but requires additional infrastructure setup.

The final choice depends on performance testing results and practical constraints.

6.2 User Interface

The system provides a chat-based interface where users:

- Type symptom descriptions in natural language
- Enter patient demographics (age, sex, illness nature)
- Receive a diagnosis suggestion with explanation
- See which historical cases informed the suggestion

6.3 System Updates

One advantage of RAG is easy updates:

- New patient records are preprocessed and embedded
- Embeddings are added to the vector database
- No retraining required—new cases immediately become searchable

This allows the system to stay current with quarterly data updates from the health department.

6.4 Operational Considerations

Monitoring: Track system performance, response times, and any errors in the workflow.

Data Security: Implement appropriate access controls and encryption, even though data is anonymized.

Version Control: Maintain records of database versions and workflow configurations for troubleshooting.

7. Limitations and Ethical Considerations

7.1 System Limitations

This is a decision support tool, not a replacement for clinical judgment. The system:

- Cannot perform physical examinations
- Cannot order or interpret laboratory tests or imaging
- Only knows about diseases present in its training database
- May struggle with rare or unusual presentations
- Reflects the patient population in Albuquerque, which may not generalize to other regions

7.2 Potential Biases

The system's performance depends on its training data. If certain demographic groups or diseases are underrepresented in the database, the system may perform poorly for those cases. Regular evaluation across different patient subgroups is necessary to identify and address disparities.

7.3 Appropriate Use

Clinicians should use this tool as one input among many in the diagnostic process. The system's suggestions should be verified against clinical findings, patient history, and additional testing. Final diagnostic decisions remain the clinician's responsibility.

8. Conclusion

This project demonstrates a practical application of retrieval-augmented generation for clinical decision support. By storing patient cases in a searchable database and using AI to reason about similar historical cases, the system provides transparent, explainable diagnostic assistance for respiratory diseases.

The two-pipeline architecture separates offline data preparation from real-time querying, enabling efficient operation and straightforward updates. Unlike traditional machine learning approaches requiring extensive retraining, this RAG-based system adapts to new data simply by adding it to the vector database.

The project represents an accessible entry point into intelligent medical systems, focusing on core RAG concepts: semantic embeddings, vector similarity search, and AI-assisted reasoning over retrieved context. While the system has clear limitations and requires careful validation, it demonstrates how modern AI techniques can augment clinical expertise in resource-constrained settings.

Success will be measured by the system's ability to correctly suggest diagnoses based on symptom input, provide understandable explanations, and maintain reliable operation. Beyond individual diagnostic queries, the system infrastructure enables analysis of disease patterns and trends, supporting broader public health monitoring objectives.